Achieving Software Quality Excellence Through Software-Hardware Co-Design and Co-Verification

Monica Bao, Sophia Lee, Havish Sripada, Joanna Mei
revampedrobotics@gmail.com

Abstract

The future of technological advancement and complexity increasingly emphasizes the importance of software-hardware co-design and co-verification, a process currently practiced in the field of embedded systems, especially robotics. This paper details the experiences of a FIRST (For Inspiration and Recognition of Science and Technology) Tech Challenge (FTC) robotics team, RevAmped Robotics, in implementing various software-hardware features to enhance a robot's user-controlled and autonomous performance.

This paper presents software-hardware co-design and co-verification methodologies to improve robot performance, usability, and optimization. The team strategically incorporated a diverse array of independently operating sensors to automate the robot and enhance its functional capability in both the user-controlled and autonomous periods. The sensors, built into different robot components, are processed with parallel executing software threads, each of which transmits the information to a different color on an LED panel. This allows the robot to communicate its state to the user in real time, allowing for simple testing of robot features and simplifying user-based decision-making. The team also successfully synthesized complex, self-adaptive localization techniques and path-generating algorithms such as Pedro Pathing to optimize the robot's automated performance. In addition, using an automated tester program allowed the team to quickly identify, isolate, and correct any sensor, motor, or software errors the robot encountered. The team improved the software quality of the robot by accurately controlling its components, communicating its state to the user in real time, optimizing the automated movements, and having an automated testing system. This is a significant step forward in demonstrating the importance of software-hardware co-design and co-verification in improving the software quality of any embedded system.

Biography

The authors are high school students from the FIRST Tech Challenge team RevAmped Robotics based in Portland, Oregon. The team has been competing for over seven years at the local and international level and represented Oregon in the 2024 World Championships. The team also aims to learn from industrial professionals to improve their engineering skills.

1. Introduction

It has long been recognized that experiential and hands-on education provides superior motivation for learning new material by providing real-world meaning to otherwise abstract knowledge. Robotics is an effective tool for hands-on learning, not only of robotics itself but of general topics in Science, Technology, Engineering, and Mathematics (STEM). Learning with robotics gives students an opportunity to engage with real-life problems that require STEM knowledge. FIRST is among the broad spectrum of avenues for pursuing robotics at the pre-university level to promote STEM worldwide. FIRST stands for "For Inspiration and Recognition of Science and Technology" and is an international youth organization focused on developing ways to inspire students in engineering and technology fields [1].

The FIRST Tech Challenge (FTC) is designed for students in grades 7–12 working in teams to compete with each other on a playing field. Teams are responsible for designing, building, and programming their robots to compete in an alliance format against other teams. The robot kit is programmed using Java. Teams, including coaches, mentors, and volunteers, are required to develop strategies and build robots based on sound engineering principles. The ultimate goal of FTC is to reach more young people with an accessible opportunity to discover the excitement and rewards of STEM.

1.1 Background for FIRST Tech Challenge Robot Programming

The average FTC season is 9 months long, with a maximum of 15 team members. The FTC competition field is 12' x 12'. Each match is played with four randomly selected teams, two per alliance. Four 18" x 18" robots must be able to navigate around each other without breaking when hit by another robot.

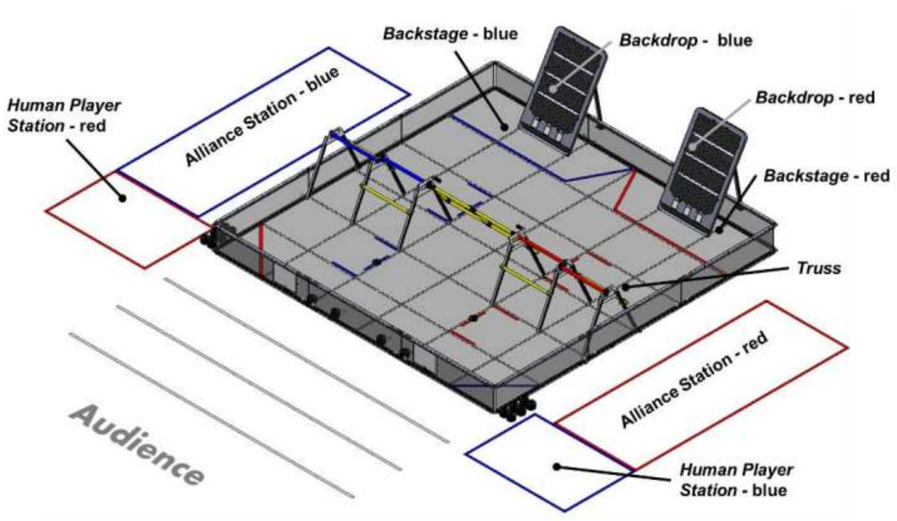


Figure 1. FTC field setup for the 2023-2024 FTC season [2]

Figure 1 shows the field setup for the 2023-24 FTC season. The FTC robot game comprises two phases: (1) the autonomous phase and (2) the driver-controlled teleoperation phase. The robot is preloaded with a yellow and purple tile called a "pixel," and a team-created "Team Prop" is placed on the field. During the Autonomous period, the robot is expected to place the purple pixel on the randomized location of the Team Element and the yellow pixel at a randomized location on the backdrop. During the driver-controlled period, the goal is to collect as many pixels as possible to deposit on the backdrop. Additional points are awarded when patterns of tiles are placed on the backdrop. During the last 30 seconds of the game, the robot is able to gain additional points by launching a paper airplane called a “drone" and fully hanging on the frame of the field.

1.2 Contributions

The FTC robot implements many advanced industrial technologies on a smaller scale, making it an excellent case study for testing many different technologies. Precise control of the robot and knowledge of its state at all times is crucial for fulfilling the many requirements of an FTC match. Robotic software must sustain accurate hardware control while presenting an intuitive interface for the user and ensuring high system performance. In addition, rapid robot improvements require an efficient and robust testing process.

The team achieved this by implementing many technologies into our robot through an LED panel display, as seen in Figure 2. The robot is acutely aware of where it is on the field, using ultrasonic distance sensors, AprilTags [10], and odometry dead wheels. Finally, advanced trajectory generations are used to improve the reliability of the robot’s automated path-following. Automated testing of the robot's components is needed for rapid iterations and maintenance during competitions.

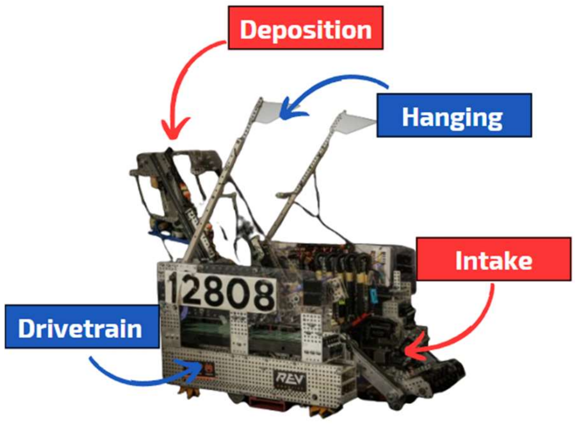


Figure 2. RevAmped Robotics’ robot for the 2023-2024 CENTERSTAGE FTC Season

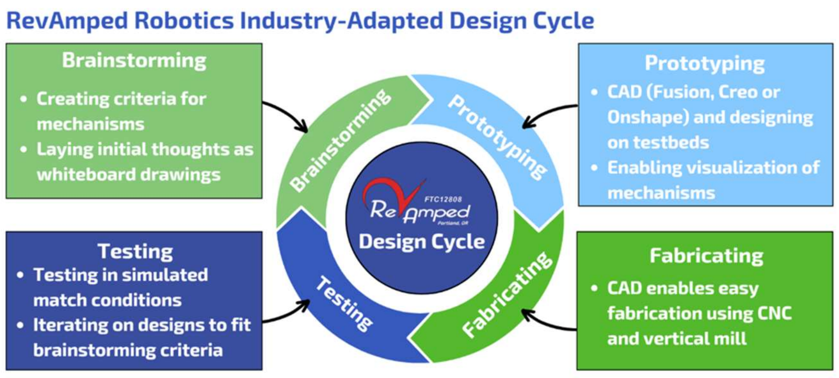
1.3 Software-Hardware Co-Design and Co-Verification in FTC robots

Software-hardware co-design [5] is the concept where both the software and hardware of an embedded system are designed simultaneously to maximize the system's capabilities. For embedded systems, trade-offs need to be made between power consumption, materials, time, complexity, and capability. Hence, designing a full system and separating the system into hardware and software components becomes essential. The same is true for all robotics.

Software-hardware co-design and co-verification [6] are pivotal in robotics due to the integration of mechanical, electronic, and computational components. Co-design involves the simultaneous development of hardware and software, ensuring that both components are optimized to work seamlessly together. This process begins with defining the robot's requirements and partitioning tasks between hardware (such as motor control and signal processing) and software (such as decision-making algorithms and path planning). After implementing both hardware and software components, co-verification techniques are employed to continue iterating upon the robot.

Co-verification ensures that the integrated hardware and software function correctly, employing comprehensive test benches, integrated testing environments, and advanced debugging tools. These methodologies' benefits include optimized performance, reduced development time, cost efficiency, and enhanced reliability and quality of robotic systems. By addressing these challenges, hardware-software co-design and co-verification enable the development of high-performance, adaptable, and reliable robotic systems capable of performing complex tasks in diverse environments.

This paper presents case studies and techniques that the team employed to follow software-hardware co-design and co-verification principles in order to build a high-functioning robot. An automated tester code quickly verifies the validity of our robotic prototypes while the team implements localization and path generation systems to increase the autonomous capabilities of our robot. Combined with sensors to monitor the state of the robot constantly, the utilized principles of software-hardware co-design and co-verification iterate upon the robot over the span of the FTC season. One strategy the team uses to employ software-hardware co-design and co-verification is our design cycle, composed of brainstorming, prototyping, fabricating, and testing. Details about the cycle are shown in Figure 3.



RevAmped Robotics Industry-Adapted Design Cycle
Brainstorming
• Creating criteria for mechanisms
• Laying initial thoughts as whiteboard drawings
Prototyping
• CAD (Fusion, Creo or Onshape) and designing on testbeds
• Enabling visualization of mechanisms
Testing
• Testing in simulated match conditions
• Iterating on designs to fit brainstorming criteria
Fabricating
• CAD enables easy fabrication using CNC and vertical mill
Brainstorming
Prototyping
Fabricating
Testing
FTC12808
ReAmped
Portland, OR
Design Cycle

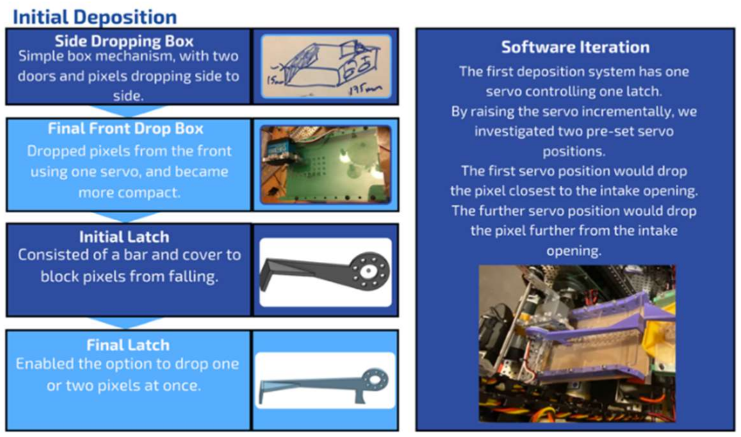
Figure 3. System Design Cycle

1.3.1 Deposition Example

One example of using software-hardware co-design and co-verification is the design of the deposition of the robot, where its finalized design was produced through two iterations of the RevAmped design cycle.

Starting with brainstorming, the team created a whiteboard drawing of their initial ideas, where they concluded that a drop-box style deposition would have the potential to be explored. Computer-aided design (CAD) plays a vital role in prototyping by allowing us to create models and simulations of each part without wasting physical materials. Each deposition design is entirely made in CAD and tested to work with other parts. To enhance our initial latch, the team wanted to prevent both pixels from depositing out at once. To do this, a stub was added to our new latch design, giving us more control while creating mosaics – a specific pattern of pixels that score additional points when created. The team decided to program different angles that the latch could be at to give us the flexibility to drop pixels onto the backdrop one at a time or all at once.

The team realized an advanced design with a more extensive range of movement was necessary. Once again, using CAD, the team designed the new parts for the deposition into two rotatable claws. After multiple iterations, we created a claw mechanism that gave us a smooth intake to deposition transfer. With 180 degrees of motion, the claw significantly increased our precision and speed when creating mosaics. This design had a total of three servos, and the use of programming software designed to synchronize the three servos allowed for a much more efficient deposition system. Figures 4 and 5 illustrate the changes in deposition system.



Initial Deposition

Side Dropping Box
Simple box mechanism, with two doors and pixels dropping side to side.

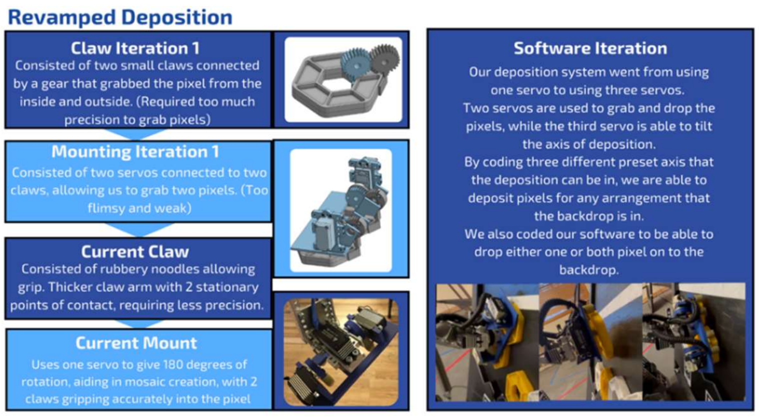
Final Front Drop Box
Dropped pixels from the front using one servo, and became more compact.

Initial Latch
Consisted of a bar and cover to block pixels from falling.

Final Latch
Enabled the option to drop one or two pixels at once.

Software Iteration
The first deposition system has one servo controlling one latch.
By raising the servo incrementally, we investigated two pre-set servo positions.
The first servo position would drop the pixel closest to the intake opening.
The further servo position would drop the pixel further from the intake opening.

Figure 4. Initial Deposition



Revamped Deposition

Claw Iteration 1
Consisted of two small claws connected by a gear that grabbed the pixel from the inside and outside. (Required too much precision to grab pixels)

Mounting Iteration 1
Consisted of two servos connected to two claws, allowing us to grab two pixels. (Too flimsy and weak)

Current Claw
Consisted of rubbery noodles allowing grip. Thicker claw arm with 2 stationary points of contact, requiring less precision.

Current Mount
Uses one servo to give 180 degrees of rotation, aiding in mosaic creation, with 2 claws gripping accurately into the pixel

Software Iteration
Our deposition system went from using one servo to using three servos.
Two servos are used to grab and drop the pixels, while the third servo is able to tilt the axis of deposition.
By coding three different preset axis that the deposition can be in, we are able to deposit pixels for any arrangement that the backdrop is in.
We also coded our software to be able to drop either one or both pixel on to the backdrop.

Figure 5. Revamped Deposition

2. Automated Testing

Automated tester code is highly beneficial for embedded systems due to its ability to provide consistency and reliability. Automated tests ensure that the robot behaves and functions correctly under a variety of circumstances to test the system's abilities. These tests require less labor than manually testing every component of an embedded system, allowing for rapid identification of problems and quick development iterations.

Automated tests are also crucial for regression testing, ensuring that new changes do not negatively impact existing functionalities. This is particularly important in complex systems where parts need to be replaced and recalibrated, as the automated tester code can catch if the new part doesn’t integrate well into the system. Furthermore, automated tests serve as documentation for the expected behavior of the embedded system's software, providing a transparent and traceable record of what has been tested and the outcomes, which aids in debugging and future development.

2.1 Automated Testing in FTC

When playing multiple games, quick robot maintenance is critical to making sure our robot is able to operate at its full potential. By having an automated tester code, our team is not only able to ensure our robot is functioning without issue, but our team can also quickly detect the core reason for any malfunction the robot is receiving. Our tester code is able to test each motor, sensor, and servo on the robot individually for functionality, allowing the team to identify if the problem faced is a hardware or software error. If the malfunction is a software error, the team will be able to identify the specific component that needs to be fixed quickly.

We are able to use our tester code for multiple functions at once. For example, two types of sensors on the robot are tested at the same time. Ultrasonic (distance) sensors and limit switches are both tested at once, which is something that cannot be done by manually testing sensors one at a time.

Sensor testing is also automated. For example, limit switches behind horizontal and vertical slides ensure that the slides don't over-retract and burn the motors powering the slides during the match. The program automatically tests the limit switches (force sensors) by extending and retracting the linear slides, measuring whether the sensor feels a change in force when the slides are away from the sensor as opposed to weighing down on it. Other sensors, such as voltage sensors and ultrasonic (distance) sensors, are also tested by the program controlling servos and motors and checking if the sensors respond correctly to this change.

Table 1. Tester Code Functions

Motors	Servos	Sensors

Drivetrain; Horizontal Slides;
Vertical Slides;
Intake Motor.
	Intake Door; Door; Intake TwoBar; Drone;
	Droppers; Hang;
	Deposition TwoBar;
	Deposition Turner;
	Deposition Claws.
		Ultrasonic Sensors;
		Voltage Sensors;
		Encoder Sensors;
		IR sensor;
		LED Panel.

This automated tester is a highly versatile tool for testing the robot quickly. Whether it's between matches during a competition, during regular maintenance of the robot, or as a confirmation that everything works as expected after making mechanical changes to the robot, it's able to identify issues and offer servo calibration tools. This eliminates the need to test every individual component of the robot manually and instead allows groups of motors, servos, and sensors to be tested at once.

3. Case Studies

3.1 Case Study: Intake Sensors + LED Panel

Through the season, each of the robot's mechanical components will undergo continuous development and refinement through a synergized software-hardware co-design and co-verification agile process. This ensures not only precision but also optimal functionality of its mechanisms. Illustrated through the exemplar of the intake system, the team has strategically incorporated a diverse array of sensors, notably leveraging the capabilities of an IR sensor to its fullest extent. Furthermore, in pursuit of enhanced operability for the robot's drivers, the team has harnessed software tools to streamline controls.

However, at the beginning of the season, our intake contained no sensors. Instead, we relied on a time-based system with pre-programmed wait-time functions to collect two pixels, the maximum amount allowed in a robot, for each cycle. While they temporarily worked, the team realized that the wait-time functions had multiple flaws. A significant issue was how the functions would sometimes cause our robot to leave the pixel stack too early or late. This led to instances where the intake would collect three or even no pixels before moving on to the following autonomous function. Such inconsistencies caused our autonomous cycles to become unreliable and resulted in a decrease in potential points. To help minimize error and maximize efficiency, the team considered using an IR sensor to detect the number of pixels in our intake.

Before incorporating the IR sensor into our robot, the team had to confirm whether or not the IR sensor would ultimately aid the robot's design more than the wait-time functions. To do this, the team utilized SWOT analysis [3] [4] to evaluate which option held the most promise. Following the acronym SWOT, the team wrote down the Strengths, Weaknesses, Opportunities, and Threats of both potential choices in two tables. After comparing the tables, the team concluded that the IR Sensor would lead to a more significant overall benefit and was mounted to the side of our intake for easy pixel detection.

Table 2. SWOT Analysis for Pre-Programmed Wait-Time Functions

STRENGTHS	WEAKNESSES
* Allows our intake to collect two pixels * Doesn't take up space in the robot	* Functions aren't always on time
OPPORTUNITIES	THREATS
* No hardware maintenance	* A lot of time spent on fine-tuning wait time * Losing points * Possibility of penalties

Table 3. SWOT Analysis for IR Sensor

STRENGTHS	WEAKNESSES
* Constantly detects the number of pixels in our robot * Very reliable * Simplify the program	* Increased power consumption * Depleted battery power quicker
OPPORTUNITIES	THREATS
* Consistently collects two pixels every cycle * Efficient autonomous cycling * Maximize points	* Not enough time to mount before the next competition

To visually compare the IR sensor with the wait-time functions, the team decided to code and run a few trials of two separate programs for the robot's autonomous phase: one for the wait-time functions and the other for the IR sensor. Figure 7 shows the results for each trial and compares how accurate each option was when intaking pixels for each trial. Looking at the graph, you'll notice that the IR sensor is much more precise and stable; every run collects two pixels, while the wait-time functions fluctuate with each trial. This not only proves that the wait-time functions are inconsistent and unreliable but also proves that the IR sensor is much more consistent.

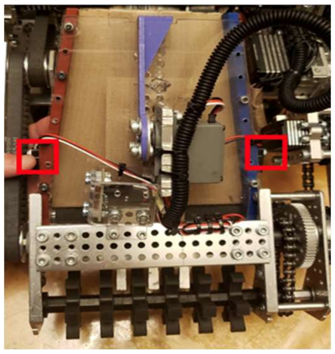


Figure 6. IR sensors on the robot's intake

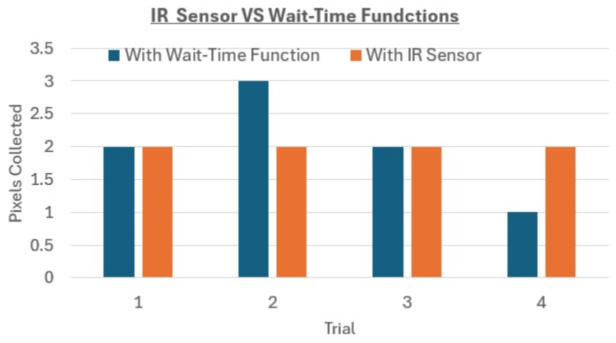


Figure 7. Comparison of wait-time functions and IR sensor accuracy

Additionally, after adding the IR sensor to the robot's design, we implemented an LED panel onto the side of our robot. Using the output from our IR sensor during the autonomous period, our LED panel lights up a specific color based on preprogrammed commands. These colors always indicate the number of pixels in our robot. This made in-game decisions for the robot's drivers more accessible and helped maximize scoring capabilities and efficiency when gathering pixels.

3.2 Case Study: Robot Movement

3.2.1 Introduction to Path-Following

Path-following has long been an essential field in the application of unmanned vehicles (UVs). The ability of such an autonomous device to determine its precise position and orientation on the coordinate field, referred to as localization, is critical to a robot's precise traversal of a given curve. This self-characterization is thus utilized in a process called trajectory generation, the process the robot uses to determine where it wants to move next. To further maximize the robot's capability, the implementation of various industry-adapted localization methodologies improved the quality of the robot's pursuit of its encoded autonomous trajectories. The exemplar of the First Tech Challenge robot, which must possess the capacity to adapt to changes in its environment, powerfully illustrates the power of software-hardware co-design and co-verification when undertaking complex tasks.

3.2.2 Mecanum Drivetrain and Odometry

In order to produce the highest accurate trajectory, robots must include the successful synthetization of both hardware and software methodologies. Flexible hardware mechanisms are critical to the robot's ability to make decisions autonomously and traverse the field in the most optimal manner. This condition is especially true in the First Tech Challenge, where robots must compete in a timed environment to rapidly navigate through their surroundings on the obstacle-dense playing field.

The team employed the mecanum drivetrain [9] to facilitate flexibility in the robot's movement mechanism. The following image reveals the various intricacies behind the mecanum drivetrain. Figure 8 shows the layout of the mecanum motors and wheels in the drivetrain. The four mecanum motors are oriented to apply force to two independent axes, allowing for precise holonomic movement [9], such as strafing.

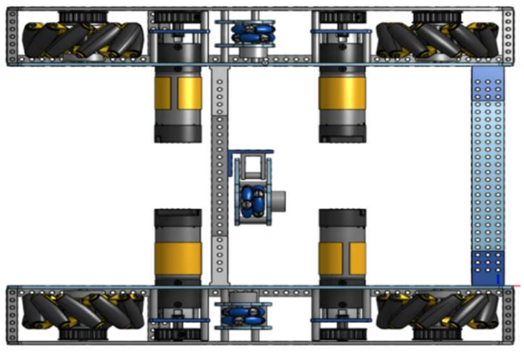


Figure 8. Bottom View of the Drivetrain Highlighting Mecanum Wheels

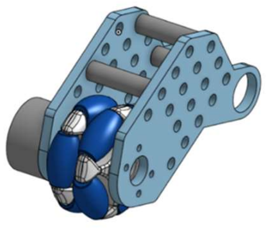


Figure 9. Odometry Pod

One example of software-hardware co-design on the robot is the three-wheeled odometry system. This odometry system utilizes dead wheels, omnidirectional wheels that do not immediately contribute to the robot's motion (as shown in Figure 9). Through these miniature dead wheels, the robot can keep track of how far it has moved. The placement of the odometry pods, which carry the dead wheels, is shown in Figure 8. Given that the competition's autonomous cycling mandated back-and-forth movement throughout the y-axis, the team decided to use two dead wheels on the y-axis and one on the x-axis to track the position most optimally. These axes refer to the field coordinate system, where the origin lies at the bottom left of the field and as the robot travels rightward its x-coordinate increases while its y-coordinate increases as the robot moves forward up the field. The coordinate system is measured in inches and spans from the point (0,0) to (144,144).

Using the odometry system, the team was able to calculate the robot's position in a looped parallel thread. Every time the robot moved, the dead wheels would track how far the robot moved by converting the number of wheel rotations to inches of movement.

3.2.3 Localization using Kalman Filter

The main co-verification localization algorithm the team implemented in conjunction with the hardware-based odometry mechanism was the Kalman filter [8], a recursive model-based methodology designed to remove noise from a system. This filter allowed the team to utilize other means of obtaining localization output and then merging those separate readings with the odometry values to create a better estimate of the robot's pose. These separate localization readings came from strategically mounted ultrasonic sensors and webcams and allowed further validation of odometry readings.

The ultrasonic distance sensors enabled measuring the robot's position on the field by measuring the distance from the robot to the field wall using the following two equations: the decision between addition and subtraction is based on which sector of the map the robot is on.

$$x = 72 \pm \max(\sin\theta, \cos\theta) \cdot ((D(S,W) + 0.5 \cdot D(R,S) \cdot D(S,E))$$

$$y = 72 \pm \max(\sin\theta, \cos\theta) \cdot D(S,W) \cdot D(S,E)$$

where $D(S,W)$ is the distance between the sensor and the wall, $D(S,E)$ is the distance between the sensor and the encoder on the y-axis, while $D(R,S)$ is the distance between the center of the robot and the sensor, and θ is the robot's current heading.

The webcam, mounted at the center of the robot's rear end, was able to provide accurate readings through the use of AprilTags, which are visual images lying above the intaking stacks and on the deposit backdrop for robots to detect. The team calibrated the webcam to utilize these AprilTags to output the robot's distance from the backdrop or stack and then applied the following equations to calculate the robot's pose, the first equation computing the robot's heading relative to the field coordinates and the following two equations computing the robot's distance from the AprilTag in either axis of the field:

$$\theta = 90 \cdot (1 + \mathrm{sgn}(D_x(C,A))) - \arctan(D_y(C,A)/D_x(C,A)) \cdot \mathrm{sgn}(D_x(C,A)) + \arctan(D_x(C,A)/D_y(C,A)) \cdot \mathrm{sgn}(D_x(C,A))$$

$$x = D_x(C,A) + D(R,C)\cos\theta \cdot \mathrm{sgn}(\theta + \tfrac{\pi}{2})$$

$$y = D_y(C,A) + D(R,C)\sin\theta$$

Where $D_x(C,A)$ and $D_y(C,A)$ represent the distance in the x and y axes from the camera to the AprilTag and $D(R,C)$ is the distance from the center of the robot to the camera. The correct equation for the heading is decided based on information from the robot's current orientation in relation to the AprilTag.

It is important to note that the distance sensors were unable to verify the robot's heading, and the algorithm would quickly fall apart if the heading estimate is inaccurate, but the webcam was able to detect the robot's heading by calculating the angle at which the robot viewed the AprilTag. This algorithm allowed the team to improve robot performance within the match further because it gave the robot a solid way to relocalize after collisions with obstacles, which would offset the odometry readings.

The importance of this algorithm can be measured through an experiment involving running the robot back and forth thrice in a straight line parallel to the y-axis. We define the error as the unsigned change of the robot's position in the axis perpendicular to movement after having terminated the motion. Figure 10 displays this error with and without the use of the Kalman Filter. The noticeable difference in errors between the trials highlights the importance of solid localization software-hardware co-verification algorithms.

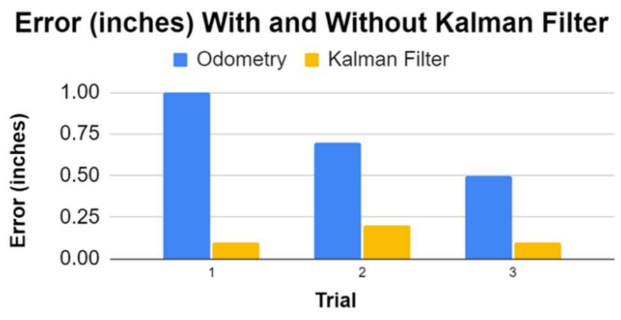


Figure 10. Comparison of Error With and Without Kalman Filter

3.2.4 Trajectory Generation

Precision and efficiency are crucial to maximizing scoring capabilities during the self-driving, autonomous period. Alongside implementing odometric localization in our robot, various trajectory planning libraries are utilized to ensure the robot achieves near-perfect path-following. To take complete advantage of the specific amount of odometry pods, the co-design capabilities of a new path-following algorithm called Pedro Pathing [7], commonly referred to as Pedro, significantly increased accuracy compared to our previously used system named Roadrunner, a motion profiling library designed for precise control over velocity and acceleration when designing complex autonomous robot trajectories.

Although the odometry pods helped increase accuracy, the critical problem with Roadrunner is that it is a motion profile-based follower. This means that a set of instructions, based on how long to run a certain magnitude of motor power, is calculated for each path before the robot begins following a trajectory. However, Roadrunner uses localization with only the purpose of generating the set of instructions for the robot's motion and, therefore, assumes that the robot is at the correct position at the end of the motion profile, changing its odometry value to match the pre-programmed target value where the trajectory was supposed to terminate. This inadequate application of software-hardware co-design means that Roadrunner is unable to correct itself after the motion profiling sequence, causing discrepancies after collisions or imprecise movement. The robot will continue to be that certain distance off from the target path for the duration of the autonomous period. This season's obstacle-dense playing field meant that substantial error in the robot's position was not sustainable.

To solve the problem of correction after collisions and imprecise movement, the team decided to utilize a more complex path planning library to create their autonomous trajectories. Pedro Pathing is a newly developed open-source system by another FTC team that improves upon the original functions of the widely used First Tech Challenge database named Roadrunner. By dynamically calculating a set of vectors that correct the robot to the closest point on the target curve before proceeding with trajectory-related motion, the Pedro Pathing algorithm allows the robot to detect and correct errors in its position after colliding with an obstacle or following inaccurate motion.

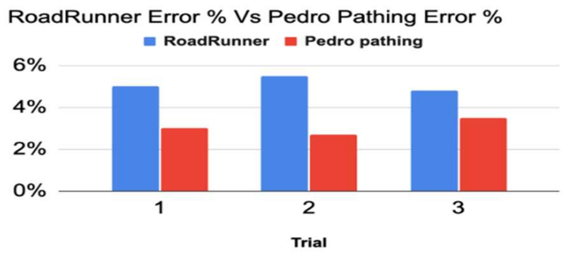


Figure 11. Comparison of the Percentage of Error for Pedro Pathing and Roadrunner

To compare the efficiency of both Roadrunner and Pedro Pathing, we developed code that moves the robot in a straight line back and forth for 30 seconds. As seen in Figure 11, multiple trials were run, and Roadrunner had an average end error of 5.1% while Pedro had an end error of 3.1%. We can notice that Pedro has a smaller margin of error and follows a lot closer to the intended path than Roadrunner. This thus proves the importance of software-hardware co-design in improving the accuracy of our robot's autonomous movement.

4.0 Data Analysis to Drive Future Development

Data analysis on past embedded systems performance, like the analysis done on our robot in Figure 12, helps pinpoint and investigate possible improvements to the embedded system, such as mechanical malfunctions or software bugs. Statistical techniques can then be used to summarize and interpret performance data, often visualized through graphs and charts, making data visualization easier for people. Strategic insights are drawn to identify areas of excellence and needed improvements, informing adjustments in strategy, design, or programming.

This process facilitates a feedback loop where findings are used to iteratively improve the embedded system's design and operational strategy. The benefits of this analysis include data-driven decision-making, a competitive edge, and continuous learning and improvement.

4.0.1 Competition Score Analysis in FTC

Over a competition season, many tournaments are played where the team receives feedback on the performance of the robot when competing with and against other robots. By tracking and analyzing competition results, long-term performance trends become evident. By examining performance across several competitions, teams can spot persistent problems or errors that might not be apparent from a single event. This signals to the team what components of the robot work well and possible improvements that could be made. Analyzing competition scores helps teams make data-driven decisions about design modifications, programming adjustments, and strategic changes.

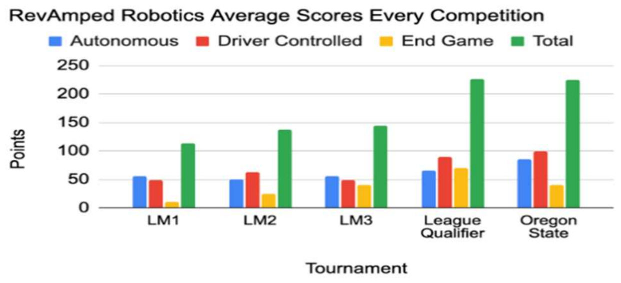


Figure 12. Average Score Per Competition

Figure 12 shows the average scores of RevAmped Robotics during every competition. The average scores of the team are broken down into points during the 3 phases of the match, the autonomous, driver-controlled, and end game periods. The total scores are also graphed.

The effects of changes to the robot can be traced back. For example, in between League Meet 3 (LM3) and the League Qualifier, we decided to add the IR sensor to the intake of the robot and have the LED panel signal the state of the intake. This drastically increased the average Driver-controlled period score of the team from 49 points to 90 points, a rise of 41 points. After the League Qualifier, the team realized that the autonomous score was stagnant. By our team switching to Pedro Pathing, an increase in the average autonomous score of 66 to 95 can be observed. Noticeable increases in robot competition performance after identifying and addressing areas of improvement for the robot show the effectiveness of the co-design and co-validation strategies employed by the team and presented in the paper.

5. Summary

This paper explores the implementation of software-hardware co-design and co-verification methodologies to enhance the performance of a robotics system, specifically the FIRST Tech Challenge team, RevAmped Robotics. The team integrated various sensors and advanced algorithms to improve both user-controlled and autonomous functions of their robot. Key innovations include the use of parallel processing for real time state communication via LEDs, self-adaptive localization techniques, and automated testing systems for rapid error identification and correction. The paper highlights the benefits of these methodologies in achieving precise control, improved usability, and optimized performance, demonstrating the importance of integrated design and verification processes in advancing the capabilities of embedded systems in robotics.

Acknowledgments

The authors would like to thank paper reviewers, Nancy Hou McCormack and Justin Wolf for their valuable assistance and feedback on software quality. Furthermore, the authors would like to thank the RevAmped Robotics team coach Zhunquin Wang for his excellent mentorship throughout our multiple-year FTC journey. In addition, the authors would like to thank the Oregon Robotics Tournament & Outreach Program (ORTOP), whose planning and assistance are crucial for the success of FTC events in Oregon. Finally, the authors would like to thank FTC team 10158 Scott's Bots for developing the Pedro Pathing software library.

References

[1] "Home." FIRST. Accessed July 31, 2024. https://www.firstinspires.org/.

[2] "Game Manual Part 2 – Traditional Events." Manchester: For Inspiration and Recognition of Science and Technology, October 11, 2023.

[3] Aha! "It SWOT Analysis (Free Templates): AHA! Software." IT SWOT Analysis (Free templates) | Aha! software, March 11, 2024. https://www.aha.io/roadmapping/guide/it-templates/swot.

[4] T Guo and M Bao. 2023. "Revamp Software Quality for Android Robots through Advanced Development and Deployment Methodologies", 41st Annual Pacific NW Software Quality Conference, Portland, OR

[5] Bailey, Brian. "Software-Hardware Co-Design Becomes Real." Software-Hardware Co-Design Becomes Real, November 6, 2021. https://semiengineering.com/software-hardware-co-design-becomes-real/.

[6] R. Mukherjee and M Purandare. 2017. "Formal Techniques for Effective Co-verification of Hardware/Software Co-designs", 54th Annual Design Automation Conference, Article No.: 35, Pages 1-6

[7] Pedro Pathing. Accessed August 14, 2024. https://pedropathing.com/Pedro+Pathing.

[8] E Kou and A Haggenmiller. 2023. "Extended Kalman Filter Estimation for Autonomous Competition Robots", *Journal of Student Research*, Vol. 12 No. 1

[9] Zeidis, K. Zimmermann. "Dynamics of a four-wheeled mobile robot with Mecanum wheels. Z Angew Math Mech" 2019; 99:e201900173. https://doi.org/10.1002/zamm.201900173

[10] "Apriltag Introduction." AprilTag Introduction - FIRST Tech Challenge Docs 0.2 documentation, 2023. https://ftc-docs.firstinspires.org/en/latest/apriltag/vision_portal/apriltag_intro/apriltag-intro.html.